A Hardware Acceleration Scheme for Memory-Efficient Flow Processing

A Hardware Acceleration Scheme for Memory-Efficient Flow Processing

Xin Yang, Sakir Sezer, Shane O'Neill
ECIT Institute, Queen's University Belfast, Northern Ireland

***Abstract*—This paper presents a hardware solution for network flow processing at full line rate. Advanced memory architecture using DDR3 SDRAMs is proposed to cope with the flow match limitations in packet throughput, number of supported flows and number of packet header fields (or tuples) supported for flow identifications. The described architecture has been prototyped for accommodating 8 million flows, and tested on an FPGA platform achieving a minimum of 70 million lookups per second. This is sufficient to process internet traffic flows at 40 Gigabit Ethernet.**

Keywords—flow lookup table; content addressable memory; hash; network flow processing

I. INTRODUCTION

In today's network systems, packets with common n-tuple information in their headers can be categorized into a single flow. These flow-based packets are manipulated at certain network elements including network adapters, routers and switches. Flow-based processing, such as flow inspection, mapping and monitoring, can be managed by a flow processor, which performs certain functions including flow lookup, identification, flow statistics, etc.

High-quality flow processing is demanded in many network security related applications, such as intrusion detection and prevention, Quality of Service (QoS) monitoring and security policy enforcement. In recent years, the development of complex programmable flow-processing platform has been motivated by emerging Software-Defined networking and cloud computing technologies. Among the aforementioned applications, flow table lookup and storage/retrieval of per-flow state information are essential functions in delivering dedicated network services.

With significant growth in network traffic and bandwidth requirements, millions of concurrent flows must be stored and searched at run-time to meet the line-rate requirements of 40Gbps and above. Operating such a function on hardware is featured as a memory-intensive design. Therefore the performance of a flow processor is highly dependent on adopted memory technologies and their bandwidth efficiencies.

Although advanced SRAM technologies, such as QDR SRAMs, are able to achieve high-speed data transfers, the memory densities of the latest QDRII+ SRAMs [1] are restricted to a maximum of 144 Megabits. In comparison, the state-of-the-art Double Data Rate (DDR) SDRAMs, in particular the third generation (DDR3) SDRAMs, are gaining popularity in the computer memory market, with large storage capability of several Gigabytes at high data-transfer rates and low-costs. There is a growing interest in building up large-scale lookup tables with DDR SDRAMs.

The biggest challenge of using DDR SDRAMs is how to overcome the table lookup/update latencies to meet packet throughput requirements in network applications. Implementing million-entrant flow lookup tables using DDR SDRAMs has become an issue in high-speed networks. In this paper, a hybrid memory-efficient method with a combination of embedded memory resource and external DDR3 SDRAMs is presented and evaluated on an FPGA device. The proposed scheme achieves resource balancing and flexibilities in high-speed and large-database flow searches and updates.

II. RELATED WORK

The basic technique existing in flow processing is to perform flow table lookup on packet headers of selected fields. Recent research on flow table lookup has been driven by stateful packet processing at wire speeds, which requires high throughputs and large table capacities.

Associative array is a fundamental data structure for fast table lookup. As a hardware solution for associative array, Content Addressable Memory (CAM) performing parallel brute-force search is commonly used for hardware accelerations on table lookup at low access latency. However due to space limitations, power issues and high fabrication costs, CAM becomes unsuitable for achieving high-capacity table lookups. Hash tables using multiple hash functions are consequently adopted as a substitute hardware solution for flow lookup to accelerate network operations.

Bloom filter is a popular approach in network applications [2]. It transforms input data into small index values using multiple hash functions. However in bloom filters, it is possible that the input data does not belong to the specific key set but a match is still found. This type of error is called a "false positive". The false positive problem of bloom filters can be alleviated by enlarging the size of the bit vector, increasing the number of hash functions, or having a small number of keys. Parallel bloom filters [3-5] have been introduced to achieve a lower false positive rate.

Alternatively, the multi-choice hashing scheme presented in [6] demonstrates equivalent speed performance to bloom filters and lower collision rate over that of conventional single hash methods. However, hash collision cannot be completely avoided and it is still an issue that must be addressed and resolved.

A pattern matching scheme using cuckoo hashing proposed by Thinh [7] is an example of an FPGA implementation using two hash functions. The drawback of this approach is the nondeterministic time to build up a hash table because the newly inserted keys sometimes need to "kick out" the keys that are already there. Once the lookup table is built, a constant O(1) lookup time can be achieved as only two locations need to be searched for each data.

Hash collisions can be effectively solved by incorporating a CAM of a reasonable size, which performs parallel searches on the colliding data. CAM-aided multi-hash tables have been exploited in recent years. The effectiveness of combining a bloom filter and a CAM was investigated by Li [8] for achieving a collision-free hash table. Kirsch [9] proposed a multi-choice hashing scheme with a small CAM of 64 entries for the overflow list. The only concern is that the additional move during insertion is impractical for high speed requirements.

A Hash-CAM concept [10] has been proven for rapid table lookups and updates. With the presence of QDRII SDRAMs, a hardware circuit proposed by Yang [11] is able to search packet headers against a 128K-entrant look up table at target line-rates. Based on the same concept, alternative memory architecture is presented in this paper to enable data lookup on large tables with several million entries. A hardware solution for the NetFlow application has been developed and prototyped on a development board with current FPGA technology.

III. PROPOSED LOOKUP SCHEME

Compared to SRAM-based Hash-CAM circuits, high random-access latency constrains the employment of DDR SDRAMs for high bandwidth search functions. In view of the fact that network traffic is formed into packets with predefined header format, the data patterns stored in the flow table are predictable to a certain extent. As commercial general-purpose DDR3 SDRAM controllers are not optimized for specific data patterns in network applications, a lookup scheme is proposed in this section to facilitate the usage of DDR SDRAM resource in high-speed flow processing.

A. Hash-CAM using DDR SDRAM

As shown in Figure 1, a two-choice hash table is used to address separate DDR memory blocks, Mem1 and Mem2, respectively. Each memory location indexed by a hash value (Hash1 or Hash2) is able to accommodate K different entries. Additional entries at the same hash location, namely hash collisions, are stored in the CAM. Any search query invokes a table lookup request in a maximum of three pipelined stages, CAM lookup, Hash1-Mem1 lookup and Hash2-Mem2 lookup, in sequence. A match occurring at any stage stops the current search and outputs a match index value. In this case, memory accesses for the subsequent stages can be omitted. If there is no match found after going through the above three stages, a memory update block (*Mem Updt*) is used to insert an entry in the table and output the corresponding location index for that entry. Unlike the conventional Hash-CAM table, in which the CAM and hash tables operate simultaneously on a request, the proposed table allows subsequent searches to be processed ahead of time if the current search completes at an earlier stage.

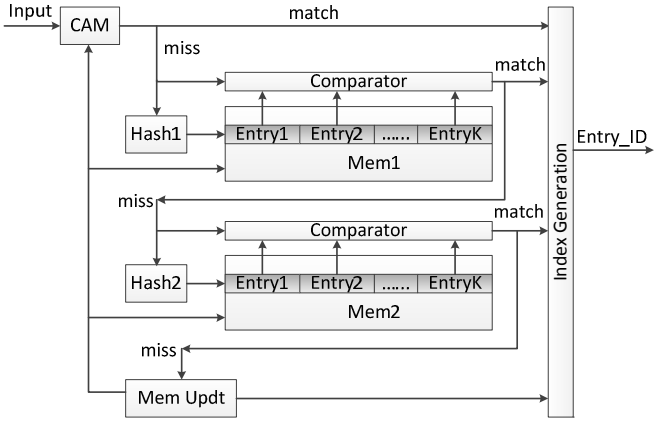

Figure 1: Hash-CAM table using DDR SDRAM

B. Flow Lookup Scheme

Based on the proposed Hash-CAM table, a flow lookup scheme that is suitable for the usage of DDR SDRAM is presented in Figure 2. In this scheme, a flow lookup table (*Flow LUT*) is split into two equivalent parts, either of which is indexed by a hash output, as represented in Figure 1. The lookup, comparison and update control logic is duplicated to form two symmetric lookup paths, path A and path B. Each path is associated with a data lookup unit (*DLU*) connecting to a separate DDR3 memory set, a *Flow Match* block and an update control (*Updt*) block. To emphasize the key idea of this scheme, components such as CAM and hash are not drawn in Figure 2.

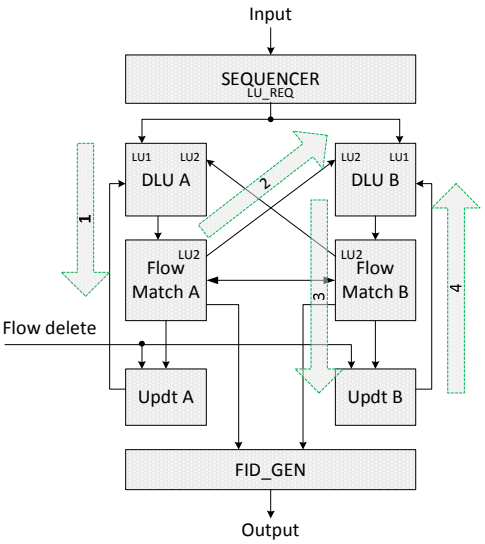

Figure 2: Proposed flow lookup scheme

For any incoming packet, a packet descriptor with n tuples (e.g. Destination/Source addresses, Destination/Source ports and protocol) is extracted from the packet header and hashed using two pre-selected hash functions. Both the original tuples and hash results are fed into a sequencer, which consists of a load balancer determining the path (A or B) that the data should go through first. On the chosen path, data from the location indexed by the hash output are read out by *DLU* from the corresponding DDR3 SDRAMs and passed into *Flow Match*. The *Flow Match* block compares all read data against the original tuples in order to find a match. In case of mismatch, the data are redirected to the other path for a second lookup. This is illustrated with four steps as illustrated in Figure 2. If a match occurs, the second lookup is omitted. Memory read/write requests are managed in the *DLU* and the *Updt* blocks for achieving better bandwidth efficiencies. A flow identification (*ID*) value is created in the *FID_GEN* block based on the search result. In the best case when both requests return matches from the first lookup stage, this proposed scheme is able to process two lookup requests simultaneously on either path and output two flow IDs in sequence.

IV. Circuit Implementation

DDR SDRAM memories are good candidates for Hash-CAM based table lookup for memory-centric networking systems. DDR memories have a multi-bank architecture and read/write accesses in the DDR memories are burst oriented. This means that for a single data access at a certain address location, other data words on the programmed addresses (e.g. the same row in a bank or across different banks) can be accessed together in predefined sequences (sequential or interleaved) in continuous clock cycles. Burst mode allows fast accesses to the DDR SDRAMs in order to achieve higher bandwidths. Figure 3 gives an example of the DQ bandwidth utilizations for different numbers of read/write bursts on the same row of a bank, at a burst length of 8. The results are calculated based on Micron's DDR3-1066 (-187E) product with parameters specified in [12].

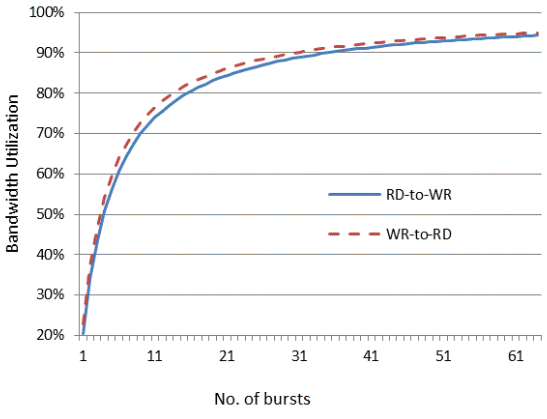

Figure 3: Continuous RD and WR bursts on the same row at BL=8

For example, if the number of bursts is 2, the access sequence to the memory is "RD-RD-WR-WR". By increasing the number of bursts from 1 to 35, the bandwidth utilization can be improved from 20% to about 90%. Because read/write transition is the main reason for discontinuous data bus usage, consequent bursts of read or write accesses for a block of data are desired for a better bus efficiency.

A. Design of the Data Lookup Unit

The data lookup unit (*DLU*) is designed to schedule read/write accesses to DDR SDRAMs. In the proposed scheme, *DLU* is the core block, which affects the overall system performance. A high-level block diagram of the proposed *DLU* and its connections is shown in Figure 4. A *DLU* consists of three blocks, a Bank Selector (*Bank Sel*), a Request Filter (*Req Filter*) and a Memory Control Logic (*Mem Ctrl*).

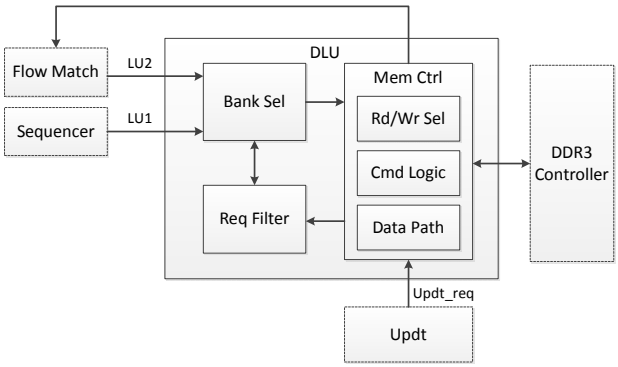

Figure 4: Data lookup unit block diagram

As depicted in Figure 2, *DLU* is used to handle two lookup requests, *LU1* and *LU2*. *LU1* comes from the sequencer at packet arrival. *LU2* comes from the *Flow Match* block on the other path in case the *LU1* on that path does not return a match. *Bank Sel* shown in Figure 4 is therefore used to queue up the two types of incoming requests and order them based on the bank information in the DDR SDRAM that they intend to access. *Req Filter* is included to manage the proceeding requests and group certain requests into a waiting list if necessary. This is to avoid the corner cases, for instance if one request is updating the memory while another request is trying to access the same location. Lookup requests and update requests including entry insertions and deletions are managed in the *Mem Ctrl* block. The proposed *DLU* can be used together with a standard DDR3 memory controller (the DDR3 Controller block in Figure 4). It pre-processes and reorders memory requests before they are sent to the memory controller. It should be noted that, this block performs flow reordering to improve memory bandwidth efficiency. The packets belonging to the same flow are still strictly maintained in order.

B. Design of the Update Block

The update block (*Updt*) consists of a request arbitrator (*Req_Arb*) and a burst write generator (*BWr_Gen*), as shown in Figure 5. Inputs to *Req_Arb* are classified into two types of

requests, deletions (*Del_req*) and insertions (*Ins_req*). *Del_req* is signaled by the housekeeping function in the *Flow State* block, which periodically checks and removes timeout flow entries to allow new flow entries to be stored. *Ins_req* is asserted by the *Flow Match* block within the *Flow LUT* if the incoming search query does not return a match. *Req_Arb* schedules the input deletion/insertion requests and forwards them as update requests in an optimized sequence. *BWr_Gen* handles the update requests sent from *Req_Arb* and instructs *DLU* with a burst of update requests. The basic idea behind *BWr_Gen* is to monitor both the time gap since the last update and the number of ongoing update requests, in order to issue burst write requests at timeout or at the time when the request count reaches the target limit.

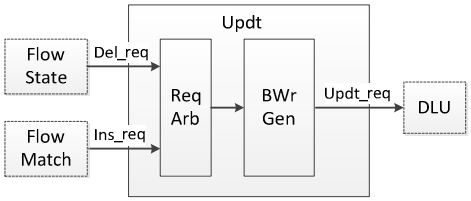

Figure 5: Update block diagram

C. FPGA Prototyping

The proposed *Flow LUT* block together with dedicated DDR3 memory controllers has been fitted into an Altera's Stratix V FPGA (5SGXEA7N2F45C2). The on-chip resource usage is listed in Table I.

Table I: Resource usage on an Altera device

Stratix V (5SGXEA7N2F45C2)	
Logic utilization (in ALMs)	31,006 (13%)
Block memory bits	2,604,288 (5%)
Total registers	39,664
Total PLLs	2
Total DLLs	2

Two quarter-rate memory controllers (DDR3 Uniphy IP from Altera) are applied to provide the user interface of read/write operations to the memories. Timing analysis indicates that the worst-case maximum clock frequency (Fmax) of *Flow LUT* is restricted to 214.59 MHz. Therefore, the implemented *Flow LUT* circuit is able to operate at the target system clock frequency of 200 MHz. Providing two separate 32-bit wide 512-MByte DDR3 SDRAMs working at the memory I/O bus clock frequency of 800 MHz, a lookup table with 8 million flow entries is implemented.

V. Evaluation & Discussion

In DDR SDRAMs, the worst-case read access latency is represented by row cycle time for successive read accesses to different rows of a bank. In addition, bus turnaround penalty is another issue, which limits the memory bandwidth efficiency. With an optimized load balancer, high bus efficiency can be achieved in the *Flow LUT* by swapping banks and grouping memory read/write access requests in order to issue long bursts of reads or writes to the memory. With this in mind, the implemented circuit was tested and verified with designed packet descriptors to address expected functionalities and corner-case conditions in different test scenarios.

A. Performance Evaluation

Basic functional tests have been performed on data lookup, table insertion/deletion and *flow ID* generation. The data processing performance with different input data patterns is evaluated and compared in Table II. In these tests, the worst-case average processing rate for 10 thousand inputs is obtained by adjusting the input data rate in the range between 60 MHz and 100 MHz.

In Table II(A), two hash patterns are adopted as the input data to the sequencer. The tests are designed to verify the effectiveness of bank selection and load balancing related to the lookup performance. Firstly, random hash values on both paths are used to compare against the predefined hash sequences with bank addresses incremented by 1. With load balancing, the proposed circuit is able to process random hash inputs at 44.05 million descriptors per second (Mdesc/s), while the processing rate is 44.59 Mdesc/s with the other hash pattern. There is no distinct degradation of data processing rate with random hash values as the bank selector works to re-organize the input data into 8 banks in the DDR3 SDRAM. On the other hand, load balancing presents good results on the circuit processing rate. As an example, if all data passes through lookup Path B, the process rate (36.53 Mdesc/s) is slower in comparison to that (44.59 Mdesc/s) with 50% load on Path A.

Table II: Performance tests on Flow LUT

(A) With defined hash patterns

Test	Description	Load - path A	Proc. rate (Mdesc/s)
Load balance & Bank selection	Random hash	50.8%	44.05
	Unique hash with bank increment	50.0%	44.59
		25.0%	41.09
		0	36.53

(B) With defined flow descriptor patterns

Test	Description	Flow miss rate	Proc. rate (Mdesc/s)
Flow match	Search on a table occupied with 10K entries	100%	46.90
		75%	54.97
		50%	70.16
		25%	94.36
		0	96.92

In Table II(B), the impact of flow miss rate on the processing rate is tested on a *Flow LUT* occupied with 10K entries. Flow descriptors were generated with standard 5-tuple format. The processing rates are tested by asserting another 10K input set with randomly distributed matched data at predefined match rates. It proves that the lower the flow miss rate during lookup, the higher the processing rate that can be achieved.

B. Discussion

For general analysis of flow processing, a minimum Layer 1 Ethernet packet size of 72 bytes is assumed (in conformance to the IEEE 802.3 standard). At 40Gbps Ethernet link, the packet processing rate is required to be 59.52 Million packets per second (Mpps) with a standard interframe gap (IPG) of 12-byte time. If the IPG is reduced to 1-byte time in the worst case, the packet processing rate is required to be 68.49 Mpps. In our experiments, as depicted in Table II(B), when flow miss rate is less than 50%, a processing rate in excess of 70 Mpps can be achieved. In fact, in flow processing operations, only the first packet of a new flow creates an entry in the flow table. All search queries of the subsequent packets belonging to the same flow should return a match. It can therefore be expected that when the table size is reasonably large, the flow miss rate is less than 50%.

Figure 6 provides example packet header analysis on real traffic data. The traffic trace file with 594 million packets was obtained in 2012 from a European switch fabric. Investigations have been made by capturing a number of packet sequences and analyzing the number of distinct flow entries represented by these packets. As shown in Figure 6, 570 flows are created for a thousand incoming packets. The ratio of new flows (B) to the number of packets (A) is 57%. If the size of the packet set is increased to 10 thousand, ratio B/A is reduced to 33.81%. This value can be reduced to below 10% if the investigated packet set is sufficiently large.

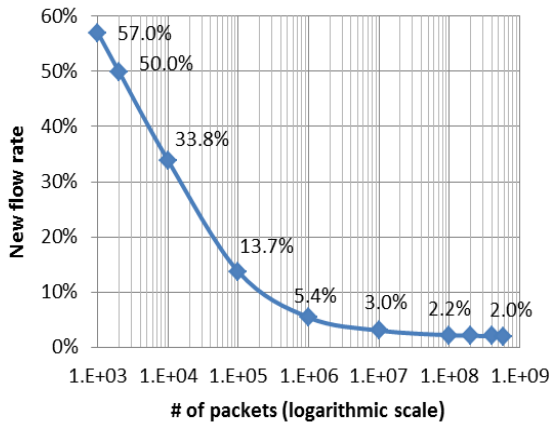

Figure 6: Real traffic packet header analysis on the selected 5 tuples

According to Figure 6, to start flow lookup with an empty table, the processing rate is expected to be slower in the initial stage because of the higher new flow rate. There is an increasing trend in the processing speed when the table size is increasing. In this work, if a lookup table has been built and maintained with 8 million concurrent flow entries, the following flow lookup hit rate is no more than 2%. At this rate, flow processing capabilities of over 94 Mdesc/s are expected, as shown in Table II(B). This enables a network throughput of over 50 Gbps with minimum packet size of 72 bytes as mentioned earlier. This achievement is competitive to the existing flow processing products, such as Cisco Catalyst 6500 Supervisor 2TXL engine presented with only 1M flow entries [13] and Netronome flow processor NFP3240 [14] with 8M flow entries but operating at a lower Ethernet link speed of 20 Gbps.

C. System Integration

Based on the prototype of a *Flow LUT*, a traffic analyzer as depicted in Figure 7 will be integrated in our test lab. The entire system targets a Stratix V Advanced Systems Development Kit with two FPGAs and 3072 MB DDR3 SDRAMs connected to each FPGA device. The proposed flow processor together with other auxiliary circuits, such as packet buffer, event engine and stats engine, will be programmed onto one FPGA. The other FPGA is reserved for packet processor performing packet payload checks such as deep packet inspection. The proposed system provides a complete solution for real-time network traffic analysis. In this system, the DDR3 memory resource is sufficient to accommodate 8 million entries while storing 512-bit per-flow related information, operating at the clock frequency of 200 MHz.

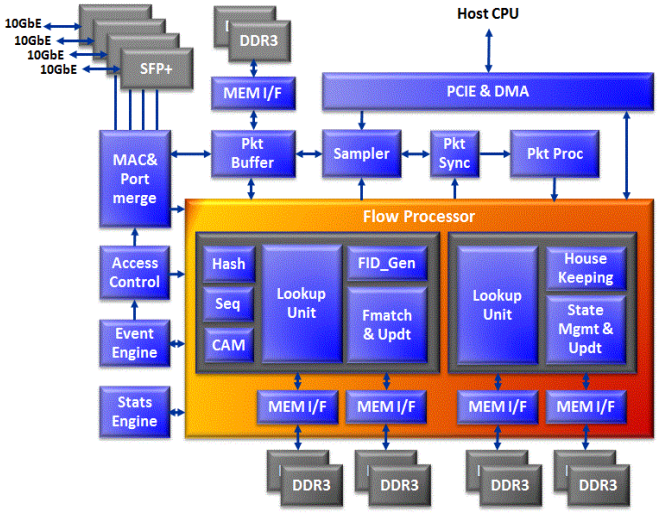

Figure 7: Undergoing system integration of a traffic analyzer

VI. Conclusion

In this paper, a novel table lookup scheme based on DDR SDRAMs is explored to deal with the rapid growth of internet traffic. With two lookup paths presented in the hash table together with advanced memory access scheduling techniques, DDR SDRAM access latency can be compensated. The solution is proposed for the target Ethernet link rate of beyond 40 Gbps. Future work includes testing on an entire flow processing system with the presented *Flow LUT* and to further improve system performance. For example, Software-Defined load balancing can be used in order to process different traffic patterns in different scenarios. A multi-path multi-hashing lookup could be considered to replace the current dual-hash scheme, for operating at a higher Ethernet link rate. The system is scalable with respect to flow table entries and number of tuples for lookup, offering high-throughput large-scale solutions for memory-rich systems.

REFERENCES

[1] http://www.cypress.com/?id=108, accessed on 6th March 2014.

[2] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey", *Internet Mathematics*, Vol. 1, No. 4, 2002, Page(s): 485 – 509.

[3] D. Sarang, K. Praveen, T.S. Sproull and J.W. Lockwood "Deep packet inspection using parallel bloom filter", *IEEE Micro*, Vol. 24, Issue 1, 2004, Page(s): 52 – 61.

[4] B. Xiao and Y. Hua, "Using Parallel Bloom Filters for Multiattribute Representation on Network Services", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 21, No. 1, Jan. 2010, Page(s): 20 – 32.

[5] H. Yu and R.N. Mahapatra, "A Power and Throughput-Efficient Packet Classifier with n Bloom Filters", *IEEE Trans. On Computers*, Vol. 60, Issue 8, 2011, Page(s): 1182 – 1193.

[6] Y. Azar, A. Broder, A. Karlin and E. Upfal, "Balanced Allocations", *Proc. of the 26th ACM Symposium on the Theory of Computing*, 1994, Page(s): 593 – 602.

[7] T.N. Thinh, S. Kittitornkun, and S. Tomiyama, "Applying Cuckoo Hashing for FPGA-based Pattern Matching in NIDS/NIPS", *ICFPT 2007*, Dec. 2007, Page(s). 121 – 128.

[8] Yun-Zhao Li, "Non-collision Hash Scheme Using Bloom Filter and CAM", *Proc. of WMWA'09, 2009*, Page(s). 55 – 58.

[9] A. Kirsch and M. Mitzenmacher, "The Power of One Move: Hashing Schemes for Hardware", *IEEE/ACM Trans. On Networking*, Vol. 18, No. 6, Dec. 2010, Page(s): 1752 – 1765.

[10] Y.-S. Chu, P.-F. Lin, J.-H. Lin, H.-K. Su and M.-J. Chen, "ASIC Design of Fast IP-Lookup for Next Generation IP Router", *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS)*, Vol. 4, May 2005, Page(s): 3825- 3828.

[11] X.Yang and S. Sezer, "Implementation of a Network Flow Lookup Circuit for Next Generation Packet Classifiers", *IEEE SOCC 2012*, Sept. 2012, Page(s): 208 – 212.

[12] http://www.micron.com/-/media/Documents/Products/Data%20Sheet/DRAM/DDR3/1Gb_DDR3_SDRAM.pdf, accessed on 6th March 2014.

[13] http://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/white_paper_c11-652021.html, accessed on 6th March 2014.

[14] http://www.netronome.com/product/flownics/, accessed on 6th March 2014.